CHAPTER 19

Active Directory Services Fundamentals

In This Chapter

Understanding the purpose behind Active Directory

Defining the Active Directory database structure

Working with the Active Directory Services Interface (ADSI)

Active Directory is one of the most exciting new technologies provided with Windows 2000. This is the feature that many network administrators have wanted for a long time because of the ease of management that a directory of this kind can provide. Active Directory also places Microsoft back on par with Novell and other competitors that have introduced their own directories. (Many people say that Microsoft still has a long way to go to catch up with competitors such as Novell because Novell has had a directory out on the market for a considerably longer amount of time.)

Whether you're a programmer, an administrator, or a user, Active Directory has something to offer you. From the user perspective, it means better support without constantly asking for assistance. All of the essential information about the user's setup is now available on a network-wide basis, not just at the user's normal workstation. The administrator's reason for liking Active Directory is the easiest to understand. Administrators can manage an entire network from one workstation now — the one located at their desk. In addition, the administrator no longer has to run to the user's workstation to find configuration information — it's maintained right within the Active Directory database. Finally, Active Directory gives programmers a new way to expand applications and make them more network friendly. Application settings no longer get tossed into the bit bucket every time the system experiences a failure. In addition, Active Directory makes it possible for the programmer to glean more information about the system as a whole, making self-tuning applications more realistic.

This chapter answers three important questions about Active Directory: architecture, construction, and the Active Directory Services Interface (ADSI). In the first section, we'll look at what Active Directory is all about. By now, you've heard a lot of hype about what this product provides. It's time to figure out what Active Directory actually provides and how these new features can help you as a programmer to get applications created quickly. We won't look at the programming issues involved in working with Active Directory, that topic is covered in Chapter 20. This section of the chapter spends a lot of time looking at Active Directory from an architectural perspective.

Active Directory is a database — albeit a very large and complex database. The next section of the chapter looks at the database element within Active Directory. We'll look at what the database contains and how it's put together. Knowing that Active Directory is a database is one thing, understanding how that database works is quite another.

Finally, we'll look at the ADSI. Obviously, this is an API designed to make it easier for you to access Active Directory from within your applications. Like many parts of Windows 2000, ADSI relies heavily on COM components to get the job done. I'll show how COM comes into place when working with Active Directory (although I'll discuss the actual interfaces in Chapter 20). In addition, there are many advanced database-specific elements to the API. For example, it's no surprise that Microsoft uses OLE-DB, their latest database access technology, as part of the methodology for accessing Active Directory. However, it may surprise you to learn that there are also Microsoft Exchange elements within this API. In sum, Microsoft has thrown in the kitchen sink when it comes to access technologies within Active Directory. Again, I'll cover the architectural elements within this chapter and leave the programming issues for the next chapter.

What Is Active Directory?

As previously stated, Active Directory is a massive database designed to make network management easier for everyone. However, you may not realize just how massive this database is. Everyone expects that Active Directory will hold the usual network information such as user and group security setups. In addition, after looking at Novell's Novell Directory Services (NDS) offering, you may expect Active Directory to help you manage network resources such as disk drives and printers. What we look at in this section of the chapter is a few of the things that you might not expect Active Directory to do. We'll also look at a few potential pitfalls you should consider when working with any network management system like Active Directory.

Look for Windows 2000 to appear in some new locations because of the Active Directory support it provides. For example, 3COM is currently working with Microsoft to embed Windows 2000 into its CoreBuilder LAN switch product series. This series of switches will use Active Directory and the Transcend Policy Manager to enforce policies for service, security, quality of service, and monitoring. In other words, this setup will provide the administrator with better control over how the switch works and a centralized place to manage it. Microsoft and 3COM are also working on a variety of other network strategies that involve Windows 2000—make sure you check out the 3COM Web site at `http://www.3com.com/nsc/switchjump/switchjump.html` for further details.

An Overview of the Interface

Microsoft has put a lot of effort into creating a new look for Windows 2000 in the form of Microsoft Management Console (MMC) and a series of snap-ins. If you get the idea that MMC is some form of COM technology, you're right. A snap-in is really nothing more than a component that uses MMC as a container.

This section assumes that you've already installed Active Directory. At the time of this writing, installation is a three-step process. You need to install both the Domain Name System (DNS) and Active Directory components during operating system installation. However, this doesn't make Active Directory active. Once you finish the server installation, you need to activate and configure DNS. Active Directory needs the DNS active before it can become active itself. Once you have the DNS activated and configured, select the installation link on the Active Directory page of the Configure Your Server utility. This link starts a wizard that leads you through the Active Directory configuration and activation process. Part of this process is to promote your server to a domain controller—a requirement for using Active Directory.

Like everything else Microsoft does, MMC is implemented on a grand scale within Windows 2000. In fact, you actually need the Configure Your Server dialog box shown in Figure 19-1 just to keep all of the snap-ins straight. Notice that I've selected the Active Directory option, which might make you think that all of the required configuration details are located within one MMC snap-in. As a result, be prepared to move around quite a bit as you test components on the server because few of the snap-ins are self-contained; they normally rely on services provided by one or more other snap-ins to provide a complete picture of a particular service or programming scenario.

The Configure Your Server utility automatically starts when you get your server configured and remains visible unless you uncheck the Show this screen at startup checkbox on the homepage. You also can start this utility using the Start | Programs | Administrative Tools | Configure Your Server command.

Windows 2000 Configure Your Server
Microsoft Windows 2000
Configure Your Server
Home
Register Now
Active Directory
File Server
Print Server
Web/Media Server
Networking
Application Server
Advanced
Active Directory
Active Directory is already installed on this server. You can now set up and manage user accounts and group settings.
Manage user accounts and group settings.
Learn more about Active Directory.
Caution
Running the Active Directory Installation wizard will demote this server to be a member server or stand-alone server depending on what type of domain controller this is. Before you proceed, be sure there are no dependent servers connected to this server.
To demote this server
1. Click Start and then click Run.

Figure 19-1: The best place to learn where you need to go next is the Windows 2000 Configure Your Server dialog box.

Notice that Figure 19-1 contains two links. The Learn More link displays a help file containing information on how to manage Active Directory. Though there isn't any programmer-specific information in the help file, Microsoft provides checklists that help ensure your server configuration is set up properly. This could mean the difference between getting an application to work the first time, or determining that your server isn't configured properly after spending hours debugging a perfectly functional application because there's a lack of tools for detecting server-side configuration problems.

Selecting the Manage link on the Active Directory page of the Configure Your Server dialog box starts a copy of MMC with the Active Directory snap-in installed as shown in Figure 19-2. This figure shows a single-server, single-domain controller system.

Let's spend a few minutes talking about the various components of the display shown in Figure 19-2. At the top of the tree is the Active Directory root. Below this is the single domain in this tree, DataConDomain. If there were multiple domains, then there would be multiple entries at this level of the tree. Don't confuse the domain controller with the domain as a whole. A domain can have more than one domain controller; these controllers appear in the Domain Controllers folder. The Builtin folder contains all of the built-in groups for the domain. The Computers folder holds a list of all the computers that have logged into the domain. Active Directory automatically manages these first three folders: Builtin, Computers, and Domain Controllers. Normally, you don't need to add new entries, but you have to configure the ones that Active Directory adds for you.

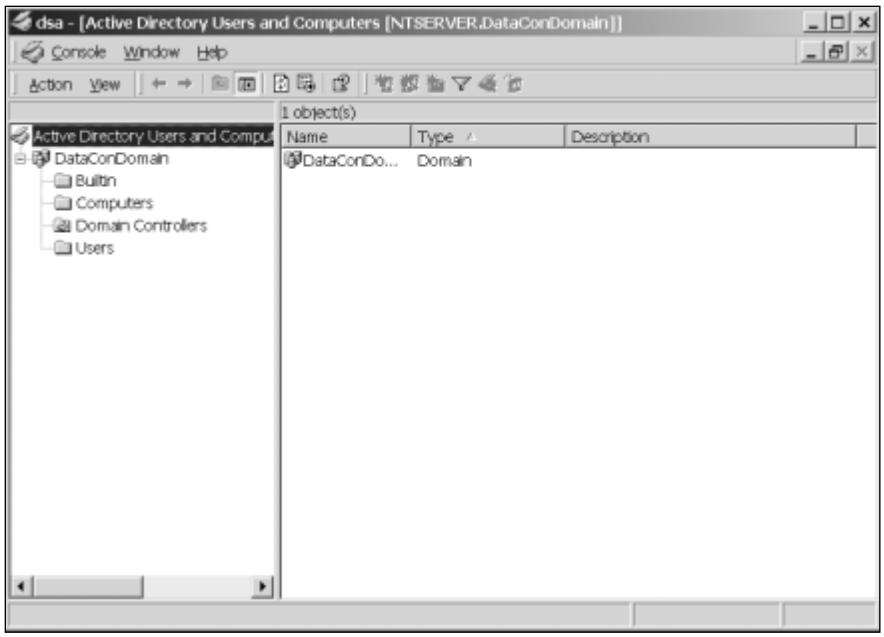

Figure 19-2: The Active Directory snap-in allows you to see the entire network at a glance, but it doesn't allow you to manage some network details directly.

The last folder, Users, is misleading because it can contain a lot more than just users. This folder can actually contain computers, contacts, groups, printers, users, and shared folders as a minimum. An administrator can create other classes of objects to add to this folder, and you can design components to work with these classes. For the most part, this is the folder you spend the most time in unless you create additional folders of your own. Active Directory allows you to add new entries at every level of the database, including the domain level. At the domain level, you can add computers, contacts, groups, organizational units, printers, users, and shared folders. However, unless you want a really messy, hard-to-follow directory, you should limit the entries at the domain level to organizational units.

One of the best ways to check whether your computer has logged into the domain and exchanged the proper information is to look in the Computers folder. The client machine appears in this folder automatically if the logon is accomplished successfully and the client and server have exchanged information. This information exchange is essential for many kinds of COM-related activities. For example, you can't even install MSMQ properly until the client has fully logged into the server and exchanged information. This is one of the first places I check now to look for problems when a component that should work (or works with other workstations) doesn't.

Developers often need to check the status of the machines that they're working with. For example, you may want to ensure that the operating system is up to date. At one time, a developer needed to keep several machines sitting on one desk so that runs between a server and workstation wouldn't be necessary. Now, a developer can use Active Directory to check on the status of the machine from a remote location. If you go into either the Computers or Domain Controllers folder and double-click a computer icon, you see a dialog box very similar to the one shown in Figure 19-3. This dialog box allows you to check the fully qualified domain name of the computer, the version of the operating system installed, what access rights the machine (not the user of the machine) has, its physical location within the company, and who manages the machine. Most of this information is entered automatically. Some information, like the computer manager, is entered manually by the network administrator, so it really pays to keep Active Directory up to date.

Figure 19-3: Active Directory automatically creates entries for your computer when it joins the domain.

There are times when you need better access to the machine than the computer Properties dialog box provides. For example, you may need to know what hardware is installed on the machine and the status of the drivers. This information is also available from Active Directory. All you need to do is right-click the computer of interest and choose Manage from the context menu. You see another copy of MMC, like the one shown in Figure 19-4. Notice that the information is grouped by System

Tools (information about the machine's hardware), Storage (the content and organization of the hard drives), and Server Applications and Services (a list of services, including COM+ and MSMQ, are available on the remote machine).

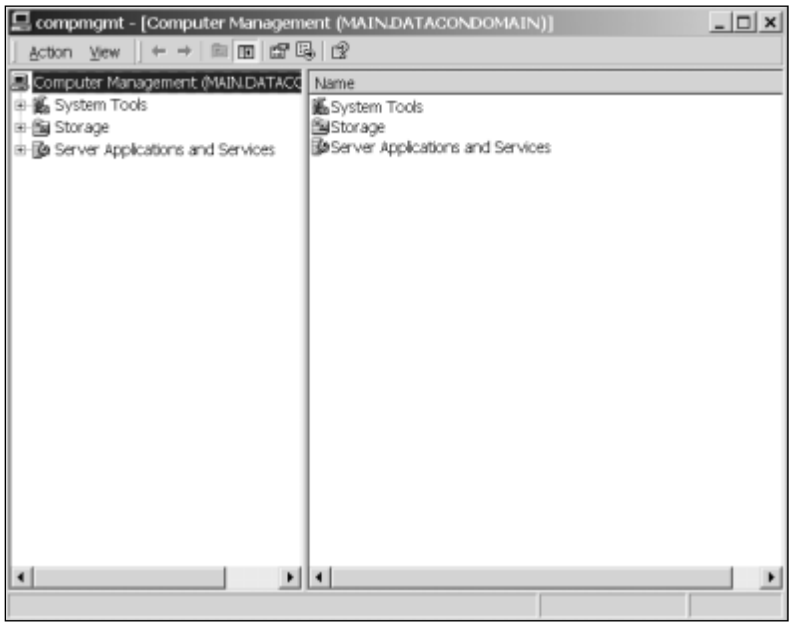

Figure 19-4: You can obtain complete information about any machine on the network through Active Directory.

Consider, for a moment, the advantages of having this information available locally rather than having to go to the machine itself. Figure 19-5 shows a typical example of the detailed information you'll see. You can use a display like this to find just the right test machine for your next project or to look for sources of problems with an application that you just wrote. A device driver or resource conflict can be causing a problem that you might not find without the services provided by Active Directory.

The displays that Active Directory provides can help you in a number of ways. For example, you can check for corruption on remote machines—an occurrence that happens all too often when users bring applications from home to use on their work machines. Instead of asking a user for input on their current configuration, help desks can check the information themselves. Your list of current drivers for the company applications allows others to do the work of finding common bugs in applications without impacting your work schedule as much.

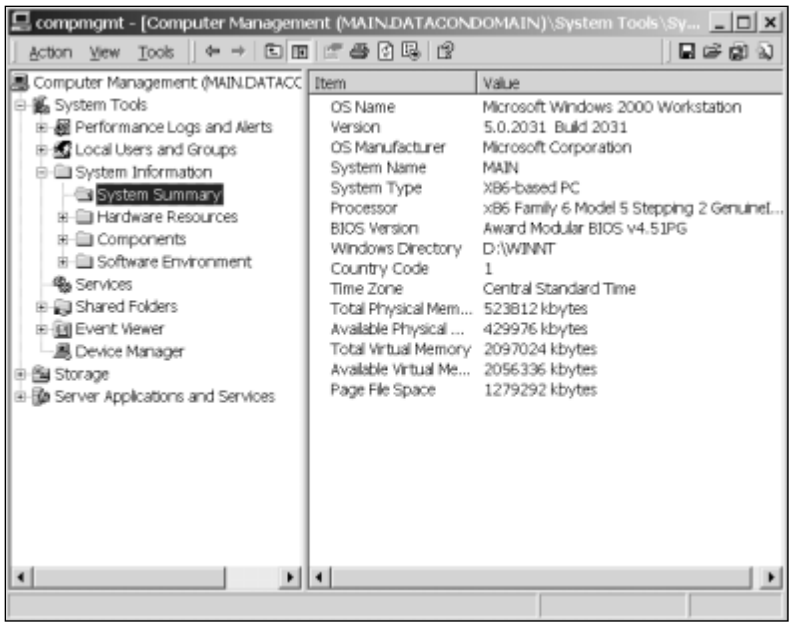

Figure 19-5: You get very complete information about each computer on the network when using the Active Directory database.

Of course, the big issue for most programmers is working with the new features that Windows 2000 provides. For example, in Figure 19-6 you can see that the client machine has MSMQ installed. All of the information that MSMQ normally stores is at your disposal, so you can check the status of queues and components to ensure an application is working as it should. In fact, you can perform this behind-the-scenes work while the user is using everyday applications. This view allows you to get a snapshot of the system as it actually functions rather than the test data you get during application design and testing. Most of us know that applications that test well in a laboratory environment can have problems when they're released for general use. Checking on the status of production machines helps you circumvent some of the issues that normally occur when an application is finally released.

By this time, you should have a better idea of why Active Directory, even the interface portion, is important for you as a programmer. Given the right tools, you can manage most testing scenarios and troubleshoot most application failures without even leaving your desk. In addition, Active Directory gives you access to real-world data, something that was difficult if not impossible to collect in the past. Users tend to behave differently when they know they're being watched. This difference in behavior affects the results you get when running tests and ultimately results in applications with less than perfect performance characteristics. Although I don't advocate a "big brother" approach to application testing, getting real-world data is an important part of working with today's complex application programming environment.

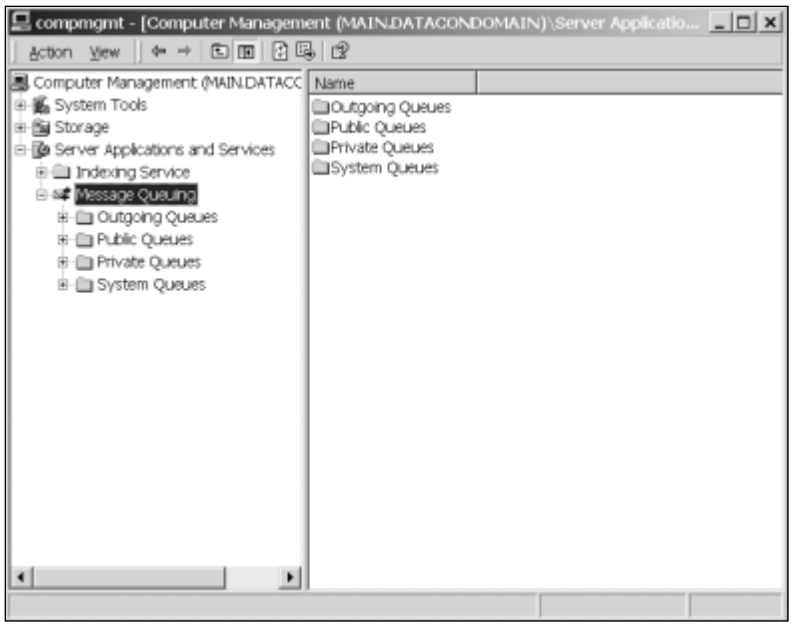

Figure 19-6: All of the features of a remote machine are available through Active Directory—even new features such as MSMQ.

Why Use Active Directory?

Active Directory has a lot to offer both the network administrator and developer alike. One of the most important considerations is that it provides complete information security. Not only does Active Directory allow you to set the security for individual objects, but you also can set security on object properties as well. This means that you can create an extremely detailed security policy that gives users access to what they need. In addition, rights at the object level can be blocked at the property level, which means that giving someone access to an object no longer means that they necessarily get full access. Finally, you can delegate the authority to manage security on an object or even a property level. I already discussed the advances that Microsoft has made in Active Directory security in Chapter 17, so I don't discuss that aspect of information security here.

Policy-based administration is another feature that Active Directory provides. Policies are really another implementation of the role-based security that I discuss for MTS in Chapter 15. Active Directory objects always have a context that defines how a particular user is using the object and what that user's rights to the object are. All of this information is stored in the Active Directory database, making it easy for an administrator to create policies that dictate the rights for entire groups of users. The combination of context, role-based security, and groups means that an administrator can look at a few groups of users rather than individual users when

managing the directory and still be sure that individual users are getting the access they require.

As a developer, you're already well aware of the extensibility that Active Directory provides. What you may not be aware of is that the administrator can extend Active Directory as well by adding new object classes or new attributes to existing classes. For example, you may want to add the number of sick leave and vacation days an employee has right to his or her entry in Active Directory. A component that you build can keep this value updated so that the employee and management can keep track of this information without having to rely on another resource. Instead of a simple contact list, you may want to create special kinds of contacts so that you can keep outside consultants separate from large customers that the company relies on for income.

Scalability is another feature that makes Active Directory a good choice. Active Directory allows you to include multiple domains in a single database. Each of these domains can have more than one domain controller. You can organize the entire setup into a contiguous namespace that Microsoft calls a directory tree. If your organization is so large that a single tree is impossible to manage, directory trees can be further combined into a noncontiguous namespace called a forest. The ability to scale a single Active Directory database over the largest organization means that when you search for a specific domain within your application, you can find it as long as you have a connection and the server is online.

So, how do all of the copies of the Active Directory database on all of those domain controllers keep in contact? Active Directory uses replication to ensure that changes made to one domain controller automatically are reflected on all other domain controllers in the entire database. This has several benefits to you as a programmer. For one thing, it means that your applications will be more fault tolerant as long as you code them to use the features that Active Directory provides. In theory, if one domain controller is overburdened or fails in some way, another domain controller can take over without any loss of functionality. Users won't even know that they're executing your application on another machine if you write it correctly.

As previously stated, DNS and Active Directory are coupled. What this means to you, as a programmer, is that domain controller and other servers could theoretically use the same name no matter how they're accessed. Users who normally access a server from their desktop computer within the company, wouldn't need to make any adjustment when accessing that same server using an Internet connection (assuming that you've properly registered your domain name). In addition, the components you create will be able to access the server in the same way no matter how the connection is made.

Active Directory can use either of two standard directory access protocols for access purposes. The most common method uses the Lightweight Directory Access

Protocol (LDAP). You can find out more about this standard method of access at http://info.internet.isi.edu/in-notes/rfc/files/rfc2251.txt. A secondary method of access is Name Service Provider Interface (NSPI). This is a Microsoft standard used with Microsoft Exchange version 4.0 and above. There are a lot of third-party products out there that work with Microsoft Exchange, so from a Microsoft-specific programming perspective, this second access method is just as important as LDAP. However, LDAP is probably the protocol that you'll use when working with multiple directory types.

The last benefit of using Active Directory is the ability to query the database using any of a number of methods. From a user perspective, you can find any object on the network using Search, My Network Places, or Active Directory Users and Computers. You discover in the next chapter that querying the database within your application is just as easy and flexible. Finding what you need isn't a problem with Active Directory.

Active Directory Programming Pitfalls

It would be frivolous to say that Active Directory will take care of every need you've ever had and will have. That just isn't realistic, despite what Microsoft's marketing arm would have you believe. There are a few ways in which a network management system such as Active Directory can actually be more of a hindrance than a help. The following list provides you with a few ideas on things that you need to keep in mind:

Domain versus Workgroup Active Directory assumes that the domain is everything and that workgroups, as such, really don't exist. Obviously, companies with more than a few employees have workgroups because this is the easiest way to work in many situations. But logging into the workgroup rather than the domain can have unexpected results. For example, you can't set up services like MSMQ without a domain connection — at least not as an independent client.

Server Loading Moving from Windows NT 4 to Windows 2000 can be quite a shock when it comes to performance. Unfortunately, the first thing that many administrators will blame is the new suite of components you've developed to take advantage of Windows 2000 features. However, a more likely culprit is all of the polling and data processing that Active Directory adds to the server. All of that information takes processing cycles and system resources to collect.

Interface Complexity	Microsoft's new MMC snap-ins may be one of the better ways to manage Active Directory, but the learning curve for this utility is astronomical, and the complexity of Active Directory doesn't help much. It seems as if there's a specialized snap-in for every occasion and a few snap-ins that replicate what the other snap-ins are doing just because it seems like a good idea. For the most part, you may find that writing applications that take advantage of everything Active Directory has to offer will increase the administrative learning curve greatly unless you can work within the confines of the current interface.
Storage	Active Directory stores everything you can imagine — and probably a few things that you don't even know exist. As a result, disk storage needs for Windows 2000 have increased greatly for the same kind of setup you may have had for Windows NT. This means that you have to exercise care when expanding the database schema or face the consequences of large disk usage increases.
Programmer Learning Curve	Everything in Active Directory relies on COM/COM+ components. A lot of developers are just learning what COM is all about, and a few may be working with their first applications. The problem is that Active Directory uses some of Microsoft's most advanced technologies, making the learning curve for developers steep to say the least. Hopefully, Microsoft will help this situation by making programming information available in a variety of languages.
New Technology	This won't always be a limitation for Active Directory, but it is today. It's unlikely that Microsoft will get everything right the first time around, so expect Active Directory, and its interfaces, to change. This means that you'll need to spend some time protecting your programming investment by using only the interfaces and methods that Microsoft has published as being complete rather than using beta or undocumented features.

As you can see, there are a lot of limitations when using Active Directory, most of which can be categorized in one of two ways. Most of the limitations you may experience are due to new resource requirements or the complexity of the Active Directory interface. It's important to keep these limitations in mind as you design projects that require Active Directory. The most important limitation for right now is the newness of the technology compared to other directory services on the market. Several years after their initial release of NDS, Novell tried to get their product completely functional and make it at least moderately reliable. One look at the Microsoft support Web sites shows that Active Directory is an even more complex product than NDS, which means that it may not be ready for production use immediately and definitely won't be ready for critical enterprise applications in the near future. However, this lack of robustness shouldn't keep you from learning about this new product and using it with smaller, noncritical applications.

Microsoft does plan to provide places where you can go for online help from peers and Microsoft support personnel. At the time of this writing, (well before the release of Windows 2000), there already are two Microsoft-sponsored newsgroups you can use: microsoft.public.active.directory.interfaces and microsoft.public.activedirectoryinterfaces.beta. As indicated by their names, both of these newsgroups are devoted to working with Active Directory using COM components. The first newsgroup tends to focus on programming issues that Microsoft has completely resolved, whereas the second tends to focus on issues that Microsoft is working on in preparation of getting Windows 2000 released.

Understanding the Active Directory Database Schema

I've discussed the fact that Active Directory is essentially a complex database manager that works with a rather large amount of data. However, I really haven't discussed how that database is put together. An understanding of the hierarchical design used by Active Directory to store information about your system is essential before you can write even a single line of Active Directory-specific code.

The first thing you need to know is the "cast of characters" in the Active Directory database world. Table 19-1 provides a quick overview of the various elements that we'll work with for the remainder of this chapter and the chapter that follows. Many of these terms will be familiar to you because you use them daily when creating applications, but they take on a slightly different meaning when used in this context. Each of these elements plays an important part in the overall composition of the Active Directory database.

Table 19-1
Active Directory Database Element Descriptions

Element	*Description*	*Purpose*
Schema	The portion of the database that defines the overall database structure.	You modify the schema every time you add a new class, modify or delete and existing class, or add, modify, or delete class attributes. In short, the schema defines the kinds of objects that you can store in the database and the characteristics of those objects. For example, if you create a user object, the schema determines whether that user object includes a telephone number.
Object	A single instance of a schema element.	An Active Directory object is similar to, but not quite the same as, the objects you create when programming. An object you create when programming is something that the user can interact with—in reality, it's part of the user's world. An Active Directory object, on the other hand, represents a real-world object. Common objects include computers, users, groups of users, and prints. All these objects have their corollary in the real world.
Attribute	An element of an object that defines what makes that object unique.	The terms *attribute* and *property* are just about the same. Active Directory objects use attributes to describe the object and to differentiate between objects of the same type. For example, a user object may have a name, address, and telephone number. These three attributes are usually enough to differentiate between two user objects. The definition of an attribute includes things such as its data type, the range of values it can accept, and any rules that you want to implement for data entry. Attributes actually exist outside the objects they define. For example, the Description attribute is defined only one time in the schema, yet it appears within most of the objects that are available as default.

Element	*Description*	*Purpose*
Class	The definition of an object that includes the object's attributes.	A class actually is part of the schema, but it's an important part. The class describes what a certain type of object will look like and how it'll be used within the database. The class is what connects the object definition and one or more predefined attributes together.
Content Rules	Defines how classes use attributes.	Attributes are divided into those that an object must have and those that an object may have given the right conditions. The content rules for a class tell Active Directory how to construct that object initially and what conditions allow the use of the "can have" attributes.
Structure Rules	Defines the hierarchical layout of the schema.	Once you get past object creation, there's still the matter of organizing everything within the schema. Classes follow a distinct hierarchy where a class that's higher than another in the hierarchy is a superclass. The topmost class, the only one that isn't inherited from any other class, is the Top class. Every other class inherits from this initial class in the schema. The structure rules define these kinds of relationships and lend order to the schema as a whole. The structure rules are always expressed by the posSuperiors attribute associated with every class within the schema.

The Active Directory database uses a hierarchical format. This means that the database is laid out in a tree configuration with parent and child objects. The hierarchical format of the schema controls what you eventually see onscreen as you work with the various database elements. Just like the real world, child objects tend to inherit attributes from parent objects, as well as incorporate a few new attributes of their own. For example, if you give users access to a shared folder, then they usually have access to all of the files within that folder unless you specially set the files to provide some other level of access.

Microsoft has already begun setting up Web sites that will help you with your Active Directory programming problems. If you want to look at Microsoft's overall strategy for building enterprise applications, then check out the Distributed interNet Applications Architecture site at `http://www.microsoft.com/dna/default.asp`. There's also a complete Active Directory programmer's guide at `http://msdn.microsoft.com/developer/windows2000/adsi/actdirguide.asp`. A technical walkthrough of various Active Directory management needs is at `http://msdn.microsoft.com/developer/windows2000/adsi/adsiwalk.asp`. In short, the information you need is available, but it may take a little time to find it. Need a multimedia walkthrough of Active Directory? You can find it at `http://www.microsoft.com/Windows/server/Demos/ActiveBasics/default.htm`. If you want a biased comparison of Active Directory and Novell's NDS, check out the Microsoft comparison site at `http://www.microsoft.com/windows/server/eval/comparisons/ADandNDScomp.asp`. Obviously, this is the Microsoft view of things and I expect Novell will come out with a comparison of their own. In some respects, these comparisons are useful because they can make you aware of features that you may not have known existed.

Active Directory stores the class definitions and attributes that make up objects within the database using the special class names of classSchema and attributeSchema. These class names are, in turn, defined as part of the directory. The reason that Microsoft went to this extreme in defining the schema elements is to allow applications to use the same LDAP operations that they normally would use to modify the schema of the Active Directory database.

Because of the way that the schema has been implemented in Active Directory, it's important to remember two important considerations when working with it. The first is that any change you make is going to affect the entire forest. This makes sense because of the replication feature that Active Directory provides. The second is that any new class you create is permanent. You can't ever delete a class once you create it, which means that you have to make class additions very carefully. Mistakes aren't reversible; there's no undo button.

Now that you have a better idea of how things are put together, there's one more element to discuss when it comes to the schema as a whole. All of these definitions and rules require some type of storage container. Active Directory provides a container in the form of the schema container. There's one schema container for every forest in which it appears. You can find out the location of the schema container by reading the schemaNamingContext of the Root DSA-specific Entry (DSE). (DSA is an acronym for Directory System Agent.)

Understanding the Active Directory Service Interface (ADSI)

Active Directory provides a lot of features that make it easier to manage large networks and safeguard the information they contain. Though the MMC snap-ins that Microsoft provides as part of Windows 2000 perform adequately when it comes to the standard classes that come with Active Directory, customized classes may require more in the way of management capability. That's why it's important that Active Directory also comes with a set of services that you can access through an application program. ADSI allows you to interact with Active Directory using a single set of well-defined interfaces.

ADSI is designed to be easy and flexible to use. For one thing, there are very few interfaces provided with ADSI right now. They're all relatively easy to understand, which means your learning curve won't be as steep as for other products currently on the market. (We look at this aspect of ADSI in the next chapter.) Microsoft intends for two completely different groups to use ADSI as a means for automating directory services tasks. Network administrators fall into the first group. Since ADSI relies on COM, a network administrator could access the features that it provides with relative ease from a scripting language. Obviously, developers fall into the other group. Microsoft hopes that developers will create Active Directory-enabled applications using the lower-level ADSI features.

Active Directory has garnered a lot of interest. A lot of this interest is from non-Microsoft sources that can help you decipher what Active Directory can mean for your organization. One of the better places to look for information about the Active Directory Server Interface (ADSI) is the 15 Seconds Web site at `http://www.15seconds.com/focus/ADSI.htm`. This site contains articles, links to other sites, a few examples, and a somewhat complete list of the more important Microsoft Knowledge Base articles for ADSI-specific topics. If you want to find out about Microsoft's view of ADSI, then check out the Active Directory Services Interfaces Overview site at `http://www.microsoft.com/windows/server/Technical/directory/adsilinks.asp`.

Now that I've introduced you to ADSI, let's take a more detailed look. The following sections help you understand what ADSI can provide in the way of programming support. We look at the actual mechanics of using ADSI in Chapter 20.

Working with a Common API

There are some advantages to using ADSI besides working with Active Directory, if you take the Microsoft approach to performing tasks. Most organizations have more than one directory service structure in place. The three most common directory services are those used by the network, e-mail program, and groupware.

If all of the directory service products in your organization conform to either the LDAP or NSPI standards, then theoretically, you could use ADSI to manage them all. I say theoretically because there are going to be a lot of situations when ADSI doesn't work simply because the vendor who created the product you're using didn't know about ADSI during development and didn't provide the required interface elements.

ADSI actually contains two levels of API support. The first level provides support for Active Directory structures that can support automation. The COM components that provide access to these structures are accessible from just about any language, as long as the language supports automation. This includes support for Java, Visual Basic, VBScript, JavaScript, and ASP. This is also the level of support that an administrator gets through a standard MMC snap-in. The second level of support is for structures that can't support automation. To gain access to this second level of support, you must use a lower-level programming language such as Visual C++.

So, how does Microsoft hope to get these third-party directory services to work with ADSI? Most of the core logic depends on LDAP or NSPI, which are fairly common standards for directory access. All a third-party vendor really needs to do is write an ADSI provider that allows directory services access through the components that Microsoft provides. That's why complete access to any directory service is only theoretical at this point—Microsoft still needs to convince those third parties to supply the required provider so that you can gain access to them through one interface.

If Microsoft does successfully write ADSI providers themselves or convince third-party vendors to perform the task, then you gain an important benefit. Any application designed to work with ADSI also works with any directory service out there. In short, you could write a single application that would allow you to manage groupware, e-mail, and the network. What this means for developers is that you'll spend a lot less time writing the same application multiple times because the directory services API for each product that your company uses will differ. The use of a single, well-defined API would reduce the amount of work required to write administration applications and free the developer up to tackle other tasks.

Creating New Objects

Like Active Directory, you aren't limited to the objects that Microsoft provides by default for ADSI. You can write new objects that extend the tasks that ADSI can perform. In some respects, this means you can write your own customized programming platform. Of course, accessing the directory database isn't a problem since ADSI supports OLE-DB. This is Microsoft's latest access technology for databases and should allow you to access quite a few of the more popular database managers.

ADSI also divides the kinds of objects that you can create into two categories. Container objects can act as objects in their own right. However, in most cases, they're used to hold other objects and allow you to interact with those objects in some consistent manner. Leaf objects, on the other hand, are standalone components designed to perform a single task.

Working with Namespaces

Every object on a network has to have a unique method of identification. Depending on the directory service that you're working with, this identification method may look similar to the method you use to access a file on your hard drive. However, most namespaces use the X.500 standard for naming things, which consists of various object type definitions, followed by that object type's value. Here's an example of what I'm talking about:

```
CN=John;OU=Editorial;O=NewMagPub
```

In this example, John is the object that we want to work with. CN stands for the context name. John is located within an organizational unit known as Editorial, which is part of an organization called NewMagPub. As you can see, this method of accessing a particular object on the network is very easy to understand and use.

ADSI doesn't support every namespace out there right now. It does, however, support four of the most common namespaces in existence: Active Directory services (ADs://), LDAP (LDAP://), Windows NT/2000 (WinNT://), and Novell Directory Services (NDS://). Notice that this namespace convention looks like a URL. You may find that the entire namespace ends up looking like an URL because of the DNS support that Active Directory provides. In fact, this is one of the reasons that Microsoft makes it so easy to find the fully qualified DNS name for the various resources on your network.

Summary

In this chapter, I discuss the theory behind Active Directory. You've learned how Active Directory can help you build a safer and more effective network. In addition, you've seen how Active Directory can reduce the network administrator's workload by providing centralized management of a wealth of resources. Even though Active Directory is relatively new, it should also help improve network reliability. Here are some additional conclusions you can draw from the contents of this chapter.

- Active Directory is essentially a large database that holds all the information required to manage a Windows 2000 network. This not only includes the usual information such as user and group security configuration, but also individual

workstation configurations as well. In fact, the administrator can manage properly configured workstations from remote locations on the network.

- Like many other parts of Windows 2000, Active Directory relies heavily on COM components for programming purposes. In this case, the COM functionality is contained within ADSI.
- One of the most important things to remember about Active Directory is that it provides standard interfaces for database management concerns as well. It uses OLE-DB to ensure that the maximum number of application types can access the directory and get any information required.
- Installing Active Directory is a three-step process. You must install the DNS and Active Directory components, configure and activate DNS, then configure and activate Active Directory.
- Active Directory provides both the administrator and the developer with access to all of the machines on the network. This feature allows the developer to perform tasks such as remote troubleshooting and real-world application testing in less time and with fewer errors.
- The reasons for using Active Directory can be summed up as information security, policy-based administration, extensibility, scalability, information replication, DNS integration, directory service interoperability, and flexible query capability.
- The two most common reasons why you need to be careful when implementing Active Directory are extensive resource use and a complex interface. In addition to these two reasons, the very newness of Active Directory when compared to other directory services on the market makes it a poor choice for large-scale, enterprise-wide, critical applications today.
- The Active Directory schema is composed of classes, objects, and attributes laid out in a hierarchical structure.
- Content rules lend structure to classes, whereas structure rules lend structure to the schema as a whole. Both sets of rules are required to provide the hierarchical structure that defines Active Directory as a whole.
- One of the more interesting features of ADSI is that it provides a common way to access multiple directory services, as long as there's an ADSI provider for that directory service. Currently, ADSI ships with providers for NDS, LDAP, Windows NT, and Active Directory.
- ADSI supports two kinds of objects: containers and leafs.
- ADSI helps cross the boundaries among the various namespaces used by directory services. It uses a common namespace for all directory services that's based on DNS.

✦ ✦ ✦